КІБЕРБЕЗПЕКА ТА ЗАХИСТ ІНФОРМАЦІЇ

УДК 004.056.55
DOI https://doi.org/10.32782/2521-6643-2024-2-68.9

Козіна Г. Л., кандидат технічних наук, доцент,
доцент кафедри інформаційної безпеки та наноелектроніки
Національного технічного університету «Запорізька політехніка»
ORCID: 0000-0002-4787-6865

Савченко Ю. В., кандидат технічних наук, доцент,
доцент кафедри кібербезпеки та інформаційних технологій
Університет митної справи та фінансів
ORCID: 0000-0002-7177-6311

Воскобойник В. О., кандидат технічних наук, доцент,
професор кафедри інформаційної безпеки та наноелектроніки
Національного технічного університету «Запорізька політехніка»
ORCID: 0000-0003-3786-8666

Прокопович-Ткаченко Д. І., кандидат технічних наук, доцент,
доцент кафедри кібербезпеки та інформаційних технологій
Університету митної справи та фінансів
ORCID: 0000-0002-6590-3898

Кацюба В. В., студент кафедри інформаційної безпеки
та наноелектроніки
Національного технічного університету «Запорізька політехніка»
ORCID: 0009-0004-4775-0172

МАТЕМАТИЧНИЙ ПІДХІД ДО ПІДВИЩЕННЯ ШВИДКОДІЇ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ КРИПТОАЛГОРИТМУ SM4

В статті запропоновано математичний підхід оптимізації алгоритму шифрування. Стаття присвячена підвищенню швидкодії обчислювальних алгоритмів, що є одним із важливих напрямків наукових досліджень. SM4 з метою підвищення швидкодії алгоритму. SM4 є досить поширеним алгоритмом, який використовується і зараз. На даний момент існує велика кількість апаратних та програмних реалізацій, в яких, використовуючи особливості архітектури чи мови програмування, можна забезпечити високу швидкодію алгоритму. Симетричні криптографічні алгоритми мають досить високу швидкодію в порівнянні з асиметричними. Симетричні алгоритми застосовуються для шифрування повідомлень або файлів великих розмірів. Нескладні етапи алгоритмів можуть зводитись до простих операцій. Такий підхід наведено в українському стандарті симетричного потокового шифрування ДСТУ 8845:2019 (алгоритм Струмок). Його особливість полягає в тому, що етапи алгоритму, нелінійна підстановка, яка виконується з використанням операції циклічного зсуву, множення члена поліному на елемент поля, спрощені до простої заміни з попередньо обчислених таблиць констант. В статті розкрито, що в результаті врахування цієї оптимізації Струмок дозволяє формувати псевдовипадкові послідовності із швидкістю понад 10 Гбіт/с. На сьогодні блоковий алгоритм шифрування SM4 широко використовується в бібліотеці LibreSSL, яка відповідає за встановлення безпечного мережевого підключення згідно протоколу SSL/TLS, та є частиною криптографічної бібліотеки Libgcrypt, яка є розробкою GNU Projects. З'ясовано, що не дивлячи на його поширеність, сам алгоритм не має шляхів оптимізації, які має Струмок. У статті здійснено оптимізацію математичного апарату SM4, в ході якої було використано особливість операції циклічного зсуву та зведено дію обчислення до прямої заміни із таблиць констант, як це було виконано в національному алгоритмі Струмок. У статті доказано, що оптимізація алгоритму представляє собою зведення обчислювальних операцій до прямої заміни з таблиць констант.

Ключові слова: *симетричне шифрування, національний стандарт, швидкодія алгоритму, криптоалгоритм SM4, оптимізація обчислювальних операцій.*

Kozina G. L., Savchenko Yu. V., Voskoboinyk V. O., Prokopovich-Tkachenko D. I., Katsiuba V. V. Mathematical approach to improving the performance of the programmed implementation of the SM4 cryptoalgorithm

The paper proposes a mathematical approach to optimizing an encryption algorithm. The article is devoted to improving the performance of computing algorithms, which is one of the important areas of scientific research. SM4 to improve the performance of the algorithm. SM4 is a fairly common algorithm that is still in use today. At the moment, there are a large number of hardware and software implementations in which, using the features of the architecture or programming language, it is possible to ensure high performance of the algorithm. Symmetric cryptographic algorithms have a fairly high performance compared to asymmetric ones. Symmetric algorithms are used to encrypt messages or large files. Simple stages of the algorithms can be reduced to simple operations. This approach is described in the Ukrainian standard for symmetric stream encryption DSTU 8845:2019 (the Stream algorithm). Its peculiarity lies in the fact that the stages of the algorithm, nonlinear substitution performed using the cyclic shift operation, multiplication of the polynomial term by the field element, are simplified to a simple replacement from pre-calculated tables of constants. The article reveals that, as a result of this optimization, Strumok allows to form pseudorandom sequences with a speed of more than 10 Gbps. Today, the SM4 block encryption algorithm is widely used in the LibreSSL library, which is responsible for establishing a secure network connection according to the SSL/TLS protocol, and is part of the Libgcrypt cryptographic library, which is developed by GNU Projects. It was found that despite its widespread use, the algorithm itself does not have the optimization paths that Strumok has. The article optimizes the mathematical apparatus of SM4, which uses the peculiarity of the cyclic shift operation and reduces the calculation action to a direct replacement from the tables of constants, as was done in the national algorithm Strumok. The article proves that the optimization of the algorithm is a reduction of computational operations to direct replacement from the tables of constants. The article describes how the tables of constants are formed, as it was done in the national algorithm Strumok. The article determines that the software implementation of the basic and optimized algorithms showed that the execution time of the optimized SM4 decreased by about 2.69 times, which means a positive impact of optimization on the algorithm's performance. The speedup was tested for different sizes of incoming messages.

Key words: *symmetric encryption, national standard, algorithm performance, SM4 cryptoalgorithm, optimization of computing operations.*

Постановка проблеми. Питання підвищення швидкодії обчислювальних алгоритмів є одним із важливих напрямків наукових досліджень. Існує безліч публікацій, наприклад [1-4], в яких описані різні підходи до вирішення цієї проблеми.

Симетричні криптографічні алгоритми мають досить високу швидкодію в порівнянні з асиметричними. Причина цього полягає у тому, що в них не використовуються складні арифметичні операції, а деякі етапи зводяться до простих перетворень (наприклад, підстановки із таблиці констант чи перестановки біт). Тому симетричні алгоритми застосовуються для шифрування повідомлень або файлів великих розмірів.

Проте нескладні етапи алгоритмів можуть зводитись до ще більш простих операцій. Такий підхід наведено, наприклад, в українському стандарті симетричного потокового шифрування ДСТУ 8845:2019 (алгоритм Струмок) [4]. Його особливість полягає в тому, що деякі етапи алгоритму (наприклад, нелінійна підстановка, яка виконується з використанням операції циклічного зсуву, або множення члена поліному на елемент поля) спрощені до простої заміни з попередньо обчислених таблиць констант. В результаті врахування цієї оптимізації Струмок дозволяє формувати псевдовипадкові послідовності із швидкістю понад 10 Гбіт/с [5].

Блоковий алгоритм шифрування SM4, створений у 2006 році, є китайським національним стандартом [6, 7]. На сьогодні блоковий алгоритм шифрування SM4 алгоритм широко використовується й за межами Китаю, зокрема в бібліотеці LibreSSL [8], яка відповідає за встановлення безпечного мережевого підключення згідно протоколу SSL/TLS, та є частиною криптографічної бібліотеки Libgcrypt, яка є розробкою GNU Projects [9]. Проте, не дивлячи на його поширеність, сам алгоритм не має шляхів оптимізації, які має Струмок.

Метою статті є дослідження і оптимізації SM4. Автори використовували переклад стандарту англійською мовою, створений у 2008 році [7]. Існують також реалізації алгоритму з використанням апаратних та програмних оптимізацій, але такі оптимізації реалізовані через особливість апаратної архітектури чи мов програмування. Тому, дивлячи на позитивний досвід алгоритму Струмок, було проведено наступне: дослідження алгоритму SM4 та його основної функції; оптимізація алгоритму через створення таблиць констант; створення програмної реалізації основного та оптимізованого алгоритмів; проведення експерименту, в якому вимірюється час виконання алгоритмів при різних розмірах вхідних повідомлень; аналіз результатів тестування.

Виклад основного матеріалу

Алгоритм шифрування SM4

SM4 – це блоковий алгоритм, розмір ключа та розмір вхідного блоку якого мають довжину 128 біт. Архітектурою шифру є сітка Фейстеля. Шифрування блоку складається з 32 раундів нелінійних підстановок [7].

Основою шифрування є перетворення T. Функція T оперує 32-бітними блоками та представляє собою суперпозицію лінійної та нелінійної замін. Функція для генерації раундових ключів (1) та шифрування блоку (2) має наступний вигляд:

$$T(a_0,a_1,a_2,a_3)=L(\tau(a_0,a_1,a_2,a_3)), \quad (1)$$

$$T'(a_0,a_1,a_2,a_3)=L'(\tau(a_0,a_1,a_2,a_3)), \quad (2)$$

де $\tau(\cdot)\tau\cdot$ – функція нелінійної заміни, L·, $L'(\cdot)$ – функції лінійної заміни.
Нелінійна заміна $\tau(\cdot)$ обчислюється за формулами:

$$\tau(a_0,a_1,a_2,a_3)=(S(a_0),S(a_1),S(a_2),S(a_3))=(b_0,b_1,b_2,b_3)=B, \quad (3)$$

де a_i – 8-бітне слово, $S(a_i)$ – підстановка з таблиці констант (S-блок, S-box).
Таблиця підстановки зображена на рис. 1.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	d6	90	e9	fe	cc	e1	3d	b7	16	b6	14	c2	28	fb	2c	05
1	2b	67	9a	76	2a	be	04	c3	aa	44	13	26	49	86	06	99
2	9c	42	50	f4	91	ef	98	7a	33	54	0b	43	ed	cf	ac	62
3	e4	b3	1c	a9	c9	08	e8	95	80	df	94	fa	75	8f	3f	a6
4	47	07	a7	fc	f3	73	17	ba	83	59	3c	19	e6	85	4f	a8
5	68	6b	81	b2	71	64	da	8b	f8	eb	0f	4b	70	56	9d	35
6	1e	24	0e	5e	63	58	d1	a2	25	22	7c	3b	01	21	78	87
7	d4	00	46	57	9f	d3	27	52	4c	36	02	e7	a0	c4	c8	9e
8	ea	bf	8a	d2	40	c7	38	b5	a3	f7	f2	ce	f9	61	15	a1
9	e0	ae	5d	a4	9b	34	1a	55	ad	93	32	30	f5	8c	b1	e3
a	1d	f6	e2	2e	82	66	ca	60	c0	29	23	ab	0d	53	4e	6f
b	d5	db	37	45	de	fd	8e	2f	03	ff	6a	72	6d	6c	5b	51
c	8d	1b	af	92	bb	dd	bc	7f	11	d9	5c	41	1f	10	5a	d8
d	0a	c1	31	88	a5	cd	7b	bd	2d	74	d0	12	b8	e5	b4	b0
e	89	69	97	4a	0c	96	77	7e	65	b9	f1	09	c5	6e	c6	84
f	18	f0	7d	ec	3a	dc	4d	20	79	ee	5f	3e	d7	cb	39	48

Рис. 1. Таблиця констант S-box

Лінійна заміна $L(\cdot)$ для шифрування та $L'(\cdot)$ для генерації раундових ключів обчислюється за формулами:

$$L(B)=B\oplus(B<<<2)\oplus(B<<<10)\oplus(B<<<18)\oplus(B<<<24), \quad (4)$$

$$L'(B)=B\oplus(B<<<13)\oplus(B<<<23) \quad (5)$$

де $\oplus$ – операція виключного АБО (XOR); $<<<$ – операція циклічного зсуву уліво.
Алгоритм SM4 має наступний вигляд:

1. На вхід подається 128-бітний ключ MK, який розкладається на чотири 32-бітних слова: (MK_0,MK_1,MK_2,MK_3). Текст повідомлення або шифртексту (назвемо InitalText, IT) розбивається на блоки по 128 біт, кожний з яких розкладається на чотири 32-бітних слова: (X_0,X_1,X_2,X_3). Якщо недостатньо байт для блоку, то такий блок заповнюється байтами нулів.

2. Генерація раундових ключів:

$$K_j=MK_j\oplus FK_j, j=0,1,2,3; \quad (6)$$

$$rk_i=K_{i+4}=K_i\oplus T'(K_{i+1}\oplus K_{i+2}\oplus K_{i+3}\oplus CK_i), i=0,1,...,31 \quad (7)$$

де $FK=(FK_0,FK_1,FK_2,FK_3)=(a3b1bac6,56aa3350,677d9197,b27022dc)$,

CK : 00070e15, 1c232a31, 383f464d, 545b6269, 70777e85, 8c939aa1, a8afb6bd, c4cbd2d9, e0e7eef5, fc030a11, 181f262d, 343b4249, 50575e65, 6c737a81, 888f969d, a4abb2b9, c0c7ced5, dce3eaf1, f8ff060d, 141b2229, 30373e45, 4c535a61, 686f767d, 848b9299, a0a7aeb5, bcc3cad1, d8dfe6ed, f4fb0209, 10171e25, 2c333a41, 484f565d, 646b7279.

3. Шифрування блоку:

$$X_i = X_{i+4} = X_i \oplus T\left(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i\right), i = 0,1,...,31 \quad (8)$$

Зашифрований блок має вигляд: Y0,Y1,Y2,Y3=(X35,X34,X33,X32) $\left(Y_0, Y_1, Y_2, Y_3\right) = \left(X_{35}, X_{34}, X_{33}, X_{32}\right)$.

Дешифрування має той самий алгоритм за винятком того, що раундові ключі застосовуються для (8) в зворотному порядку: (rk31,rk30,…rk0) $\left(rk_{31}, rk_{30}, ..., rk_0\right)$.

Оптимізація SM4

Розглянемо функції лінійної заміни (4) і (5). Елементарні операції, які використовуються у цих функціях, – це виключне АБО (XOR) та циклічний зсув вліво.

Циклічний зсув можна представити як множення матриці зсуву (L або L') на 32-бітний вектор (рис. 2, 3).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1		1								1								1						1							
	1		1								1								1						1						
		1		1								1								1						1					
			1		1								1								1						1				
				1		1								1								1						1			
					1		1								1								1						1		
						1		1								1								1						1	
							1		1								1								1						1
1								1		1								1								1					
	1								1		1								1								1				
		1								1		1								1								1			
			1								1		1								1								1		
				1								1		1								1								1	
					1								1		1								1								1
1						1								1		1								1							
	1						1								1		1								1						
		1						1								1		1								1					
			1						1								1		1								1				
				1						1								1		1								1			
					1						1								1		1								1		
						1						1								1		1								1	
							1						1								1		1								1
1								1						1		1								1							
	1								1						1		1								1						
		1								1						1								1		1					
			1								1						1								1		1				
				1								1						1								1		1			
					1								1						1								1		1		
						1								1						1								1		1	
							1								1						1								1		1
1								1								1						1								1	
	1								1								1								1						1
L0								L1								L2								L3							

Рис. 2. Матриця зсуву L

Звідси, перетворення (4) і (5) можна замінить на множення відповідних матриць.

Далі матриці L та L' представимо в вигляді конкатенації чотирьох матриць: $L = \left(L_0 \| L_1 \| L_2 \| L_3\right)$, $L' = \left(L_0' \| L_1' \| L_2' \| L_3'\right)$, кожна із яких має розмір 32×8.

Таким чином, отримуємо:

$$L\left(B\right) = L\left(b_0, b_1, b_2, b_3\right) = \left(L_0 \| L_1 \| L_2 \| L_3\right) \times \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = L_0 \cdot b_0 \oplus L_1 \cdot b_1 \oplus L_2 \cdot b_2 \oplus L_3 \cdot b_3, \quad (9)$$

$$L'\left(B\right) = L'\left(b_0, b_1, b_2, b_3\right) = \left(L_0' \| L_1' \| L_2' \| L_3'\right) \times \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = L_0' \cdot b_0 \oplus L_1' \cdot b_1 \oplus L_2' \cdot b_2 \oplus L_3' \cdot b_3. \quad (10)$$

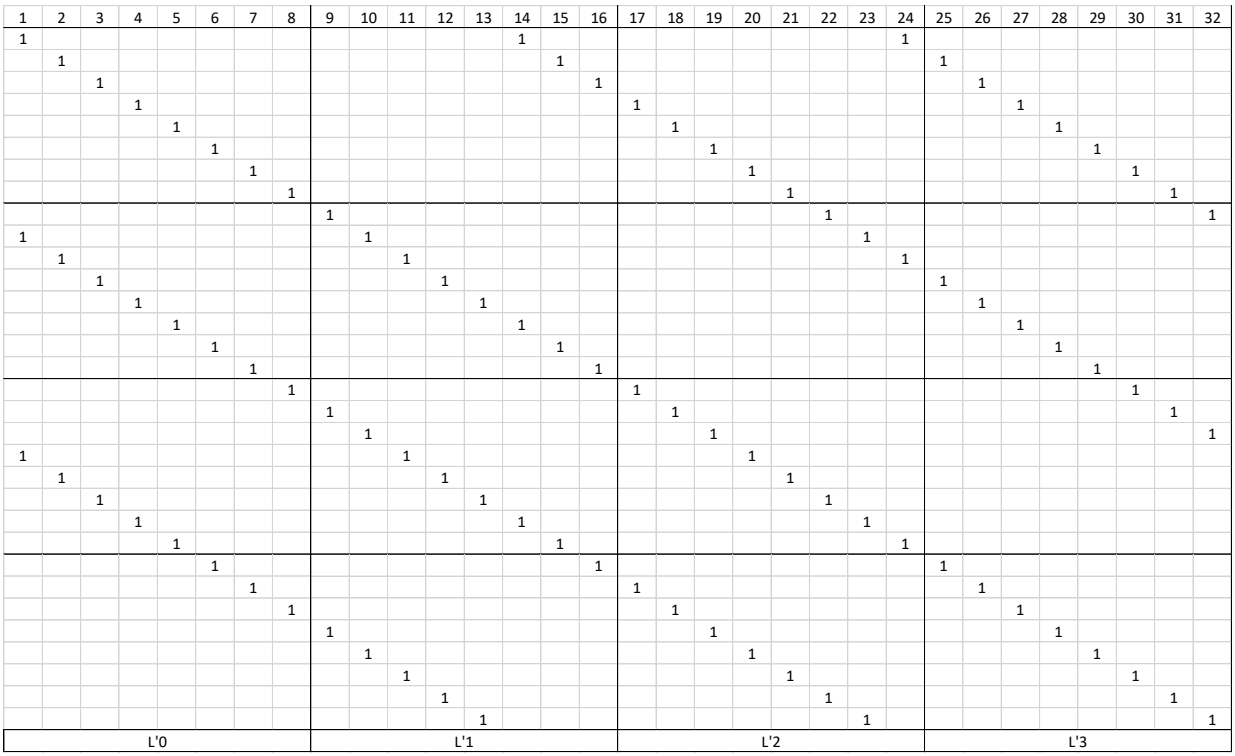

Рис. 3. Матриця зсуву L'

Тепер побудуємо матриці констант.

Візьмемо матрицю L_0. Для кожного байта a обчислимо $L_0 \cdot S(a)$. Результати обчислень запишімо в вектор A_0: $A_0[a] = L_0 \cdot S(a)$. Оскільки $a \in [0;255]$ bi∈0;255, то матриця констант $A_0[a] = L_0 \cdot S(a)$ Ai=Li·bi буде мати 256 різних 32-бітних значень. Аналогічно отримаємо вектори A_1, A_2, A_3.

Звідси обчислення функції $T(a_0, a_1, a_2, a_3) = L(\tau(a_0, a_1, a_2, a_3))$ можна замінити на відповідне значення констант із векторів A_0, A_1, A_2, A_3:

$$T(a_0, a_1, a_2, a_3) = A_0[a_0] \oplus A_1[a_1] \oplus A_2[a_2] \oplus A_3[a_3]. \quad (11)$$

Аналогічне отримаємо вектори A_0', A_1', A_2', A_3': $A_i'[a] = L_i' \cdot S(a)$. Обчислення $T'(a_0, a_1, a_2, a_3) = L'(\tau(a_0, a_1, a_2, a_3))$ також замінюємо на відповідне значення констант із векторів A_0', A_1', A_2', A_3':

$$T'(a_0, a_1, a_2, a_3) = A_0'[a_0] \oplus A_1'[a_1] \oplus A_2'[a_2] \oplus A_3'[a_3]. \quad (12)$$

В результаті оптимізації отримано чотири таблиці A_0, A_1, A_2, A_3 для шифрування блокуXAi та чотири таблиці A_0', A_1', A_2', A_3' для генерації раундових ключів rkAi. Ці таблиці надані у GitHub-репозиторії [5].

Висновки з дослідження і перспективи подальших розвідок у цьому напрямі

Програмна реалізація для дослідження приросту швидкості оптимізованого алгоритму SM4

Для вимірювання часу виконання оптимізованого та не оптимізованого SM4 було написано програму на C++, що має назву Runtime Compare. Код програми викладений GitHub-репозиторій [6]. Програма знаходиться у стані прототипу, тому функціональність програми обмежена. Для того, щоб змінити, наприклад, кількість ітерацій запуску алгоритмів, необхідно поміняти відповідне значення в коді вручну.

Програма Runtime Compare є консольною, і взаємодія з нею реалізована через файли, що знаходяться в директорії з програмою. У файл INITIALTEXT.txt записується текст, який треба розшифрувати, або шифрограма, яку треба розшифрувати. Цей файл відповідає за вхідні дані. У ході роботи програми створюються HEX_INPUT.txt, куди виводиться початковий текст у шістнадцятковому вигляді.

Програма Runtime Compare працює за наступним алгоритмом.

1. Об’явлення та ініціалізація початкової строки initialtext з файлу INITIALTEXT.txt.

2. Доповнення initialtext нуль-символами (0x00, \0) та його конвертація з символьного у шістнадцятковий вигляд; зберігання результату у файл HEX_INPUT.txt та відповідний рядок.

3. Створення потоку виводу у файл RUNTIME RESULTS.txt.

4. Цикл з COUNT_RUNS (за замовчуванням 10), де викликається функція SM4Process() з режимом оптимізації та без нього. Під час виконання кожної ітерації зберігаються результати часу виконання оптимізованого та звичайного алгоритмів.

5. Програмне закриття вище вказаних файлів.

Результатом роботи програми є файли SM4_HEX_OUTPUT.txt (туди виводиться кінцевий текст у шістнадцятковому вигляді), CT_CHECK.txt (кінцевий текст у символьному вигляді) та RUNTIME RESULTS.txt (результати середнього часу виконання програм).

Результати тестування

Для порівняння часу виконання оптимізованого та неоптимізованого алгоритмів за допомогою програми Runtime Compare було використано в якості вхідного параметра текстові повідомлення різних розмірів, починаючи з 17 байт і закінчуючи 777 кБ.

Обчислення проводились на процесорі AMD Ryzen 5600U, оперативна пам'ять 16 ГБ DDR4 SDRAM, операційна система Windows 11 Pro x64.

1) Дослідження на шифруванні повідомлення «HP Probook 455 G8» – 17 байт (табл. 1).

Таблиця 1

Дослідження часу виконання у дослідженні 1

Номер запуску програми	SM4	SM4 оптимізований	SM4/ SM4опт
1	0,005442	0,001882	2,8916
2	0,005306	0,001824	2,90899
3	0,005339	0,001779	3,00112
4	0,005415	0,001786	3,03191
5	0,005358	0,001859	2,88219
6	0,005460	0,001806	3,023255
7	0,005350	0,001782	3,00224
8	0,005601	0,001746	3,2079
9	0,005321	0,001802	2,95283
10	0,005575	0,001761	3,16581

2) Повідомлення розміру 8,17 кБ.

Таблиця 2

Дослідження часу виконання у дослідженні 2

Номер запуску програми	SM4	SM4 оптимізований	SM4/ SM4опт
1	1,044030	0,399793	2,61143
2	1,037570	0,399767	2,59544
3	1,036090	0,397134	2,60299
4	1,035880	0,398039	2,60246
5	1,034880	0,400610	2,58326
6	1,035320	0,398299	2,59935
7	1,034470	0,398316	2,59711
8	1,034280	0,398211	2,59732
9	1,041800	0,399316	2,60896
10	1,039400	0,399398	2,60242
Середнє SM4/ SM4опт			**2,60007**

3) Повідомлення розміром 777 кБ.

Таблиця 3

Дослідження часу виконання у дослідженні 3

Номер запуску програми	SM4	SM4 оптимізований	SM4/ SM4опт
1	122,768000	46,287000	2,65232
2	116,875000	56,852500	2,05575
3	121,458000	69,013800	1,75991
4	125,097000	72,430500	1,72713
5	141,115000	58,173000	2,42578
6	142,944000	59,218600	2,41384
7	129,728000	52,231900	2,48369
8	141,224000	62,063100	2,27549
9	154,018000	65,593300	2,34808
10	165,281000	57,433000	2,87781
Середнє SM4/ SM4опт			**2,30198**

Можна побачити, що приріст швидкості при оптимізації алгоритму SM4 у середньому складає в 2,69 рази, а це означає, що оптимізація SM4 є достатньо суттєвою і дає достатній приріст швидкості. Тому оптимізація шляхом зведення операції циклічного зсуву до прямої заміни з таблиць констант позитивно вплинула на швидкодію алгоритму SM4.

Висновки

SM4 є досить поширеним алгоритмом, який використовується і зараз. На даний момент існує велика кількість апаратних та програмних реалізацій, в яких, використовуючи особливості архітектури чи мови програмування, можна забезпечити високу швидкодію алгоритму. Авторами було здійснено оптимізацію математичного апарату SM4, в ході якої було використано особливість операції циклічного зсуву та зведено дію обчислення до прямої заміни із таблиць констант, як це було виконано в національному алгоритмі Струмок. В результаті математична оптимізація SM4 зменшила час виконання алгоритму приблизно у 2,69 рази, що означає позитивний вплив оптимізації на швидкодію алгоритму.

Список використаних джерел:

1. Задірака В.К., Терещенко А.М. (2021). Комп'ютерна арифметика багаторозрядних чисел у послідовній та паралельній моделях обчислень. Київ. 136 с.

2. Циганкова О. В. (2021). Методи підвищення швидкодії асиметричних криптосистем з використанням еліптичних кривих у формі Едвардса: автореферат дисертації на здобуття наукового ступеня кандидата технічних наук: спец. 05.13.21 – «Системи захисту інформації»; Київ 22 с.

3. Ткачук Р.А., Цуприк Г.Б., Яворський Б.І. (2012). Підвищення інформативності та швидкодії біотехнічних систем. Оптико-електронні інформаційно- енергетичні технології. № 2(24). С. 81-85.

4. Інформаційні технології. Криптографічний захист інформації. Алгоритм симетричного потокового перетворення. ДСТУ 8845:2019. Введ. 01-10- 2019. К.: ДП «УкрНДНЦ», 2019.

5. Кузнецов О.О., Горбенко І.Д., Горбенко Ю.І., Олексійчук А.М., Тимченко В.А. (2018). Математична структура потокового шифру Струмок. *Радіотехніка*. Вип. 193. С. 17-27. URL: http://nbuv.gov.ua/UJRN/rvmnts_2018_193_4.

6. GM/T 0002-2012 SM4 Block Cipher Algorithm (English) URL: http://www.codeofchina.com/standard/GMT0002-2012.html.

7. Whitfield Diffie and George Ledin. SMS4 Encryption Algorithm for Wireless Networks. – URL: https://eprint.iacr.org/2008/329.pdf

8. LibreSSL for Windows. SourceForge. URL: https://sourceforge.net/projects/libressl-3-2-0-for-windows/ (date of access: 04.01.2024).

9. Koch W., Schulte M. (2023). The Libgcrypt Reference Manual. 150 p.

10. GitHub: Let's build from here ·GitHub. URL: https://github.com/vityak2k22/SM4_Runtime_Compare/blob/main/Config.h (date of access: 04.01.2024).

11. GitHub – vityak2k22/SM4_Runtime_Compare: Optimization of SM4 algorithm + Runtime compare between classic and optimized algorithms. GitHub. URL: https://github.com/vityak2k22/SM4_Runtime_Compare (date of access: 04.01.2024).

References:

1. Zadiraka V.K., Tereshchenko A.M. (2021). Komp'yuternaya arifmetika bol'shikh ryadov chisel v posledovatel'nykh i paralel'nykh modelyakh opisaniya. Kiyev. 136 s

2. Tsyhankova O. V. (2021). Metody shvydkoho pidvyshchennya asymetrychnoyi kryptosystemy z vykorystannyam eliptychnykh kryvykh u formi Edvardsa: avtoreferat dysertatsiyi na zdobuttya naukovoho stupenya kandydata tekhnichnykh nauk: spets. 05.13.21 – "Systemy zakhystu informatsiyi"; Kyiv. 22 s.

3. Tkachuk R.A., Tsupryk H.B., Yavors′kyy B.I. (2012). Pidvyshchennya informatyvnosti ta shvydkosti biotekhnichnykh system. Optyko-elektronni informatsiyno-enerhetychni tekhnolohiyi. № 2(24). S. 81-85.

4. *Informatsiyni tekhnolohiyi. Kryptohrafichnyy zakhyst informatsiyi. Alhorytm symetrychnoho potokovoho peretvorennya. DSTU 8845:2019*. (2019). Vved. 01-10-2019. K.: DP «UkrNDNTS».

5. Kuznetsov O.O., Horbenko I.D., Horbenko YU.I., Oleksiychuk A.M., Tymchenko V.A. (2018). *Matematychna struktura potokovoho shyfru Strumok*. *Radiotekhnika*. Vyp. 193. S. 17-27. URL: http://nbuv.gov.ua/UJRN/rvmnts_2018_193_4.

6. GM/T 0002-2012 SM4 Block Cipher Algorithm (English) URL: http://www.codeofchina.com/standard/GMT0002-2012.html.

7. Whitfield Diffie and George Ledin. SMS4 Encryption Algorithm for Wireless Networks. – URL: https://eprint.iacr.org/2008/329.pdf

8. LibreSSL for Windows. SourceForge. URL: https://sourceforge.net/projects/libressl-3-2-0-for-windows/ (date of access: 04.01.2024).

9. Koch W., Schulte M. (2023). The Libgcrypt Reference Manual. 150 p.
10. GitHub: Let’s build from here ·GitHub. URL: https://github.com/vityak2k22/SM4_Runtime_Compare/blob/main/Config.h (date of access: 04.01.2024).
11. GitHub – vityak2k22/SM4_Runtime_Compare: Optimization of SM4 algorithm + Runtime compare between classic and optimized algorithms. GitHub. URL: https://github.com/vityak2k22/SM4_Runtime_Compare (date of access: 04.01.2024).